(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 770 957 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
02.05.1997 Bulletin 1997/18

(51) Int Cl.6: G06F 9/445, G06F 1/00

(21) Application number: 96307347.3

(22) Date of filing: 09.10.1996

(84) Designated Contracting States:
DE FR GB NL SE

(30) Priority: 26.10.1995 US 547720

(71) Applicant: SUN MICROSYSTEMS INC.
Mountain View, California 94043-1100 (US)

(72) Inventor: McManis, Charles E.
Sunnyvale, California 94087 (US)

(74) Representative:
Cross, Rupert Edward Blount et al
BOULT WADE TENNANT
27 Furnival Street
London EC4A 1PQ (GB)

(54) System and method for protecting use of dynamically linked executable modules

(57) A computer system has a program module verifier and at least first and second program modules. Each program module includes a digital signature and an executable procedure. The first program module furthermore includes a procedure call to the second procedure module, a procedure call to the program module verifier that is logically positioned in the first program module so as to be executed prior to execution of the procedure call to the second program module, and instructions preventing execution of the procedure call to the second program module when the procedure call to the program module verifier results in a verification denial being returned by the program module verifier. The second program module includes an executable procedure to be performed in response to the procedure call by the first program module to the second program module, a procedure call to the program module verifier that is logically positioned in the second program module so as to be executed prior to completion of execution of the second program module's executable procedure, and instructions preventing completion of execution of that executable procedure when the program module verifier retums a verification denial with respect to the first program module. The program module verifier responds to procedure calls by verifying the authenticity of any specified program module and by returning a verification confirmation or denial. When the program module verifier fails to verify the authenticity of a program module, the calling program module throws an exception and aborts its execution.

Printed by Jouve, 75001 PARIS (FR)

Description

The present invention relates to systems and methods for restricting the use of executable modules such that each executable module can be dynamically linked only to other executable modules whose authenticity has been verified.

BACKGROUND OF THE INVENTION

In the context of a computer program, an "external function" is typically a procedure or function located in a library or other repository of functions that is external to the computer program using the external function. External functions are often, but not always, authored by different people or by different entities than the computer programs using those external functions.

Program execution environments that allow external functions to be bound at run time, rather than at link time or compile time, facilitate the maintenance and updating of computer programs because for such programs to be used in such execution environments only the computer programs that are being revised or updated need to be recompiled, while the other modules can be left unchanged. Furthermore, the recompilation process is simplified because the compilation of the revised programs can be performed even if other modules used by the program are not present in the program development system.

However, systems using such program execution environments are vulnerable, because the interfaces between the program modules are usually well specified, or can be determined by third parties, and it is possible for such third parties to therefore use those program modules in ways not sanctioned by the corresponding software license agreements. Alternately, such third parties can allow the system to be subverted by replacing authentic program module with corrupted ones.

This problem is magnified when dealing with cryptographic routines in software that is destined for export from the United States of America to customers or distributors in other countries. It is currently forbidden by U.S. trade law to export software modules that provide general cryptographic capabilities. On the other hand, it is allowed to export programs that use cryptographic capabilities in a limited context and that cannot be used to perform general cryptographic functions outside the limited context of the exported program. In fact, it is commercially important to be able to design software systems for export that use cryptographic functions in an authorized manner. Prior art late bound systems, such as dynamic link libraries (DLLs in the Windows system) or shared objects (.so files in Solaris), attempt to solve this problem by either obscuring the interfaces between software modules, or by providing separate "export only" versions of their software. Providing separate "export only" versions of software products leads to problems in keeping the domestic and export versions "synchronized" with respect to upgrades and maintenance revisions with a single code base.

Another example of a situation where there is a need to limit or prevent use of dynamically linkable modules is an application written by a vendor that wishes to keep some functions in the application private for either trade secret or contractual reasons. Such systems require limiting access to these private functions.

SUMMARY OF THE INVENTION

In summary, the present invention is a computer system having a program module verifier and at least first and second program modules. Each of the program modules includes a digital signature and an executable procedure. The first program module furthermore includes a procedure call to the second procedure module, a procedure call to the program module verifier that is logically positioned in the first program module so as to be executed prior to execution of the procedure call to the second program module, and instructions preventing execution of the procedure call to the second program module when the procedure call to the program module verifier results in a verification denial being returned by the program module verifier.

The second program module includes an executable procedure to be performed in response to the procedure call by the first program module to the second program module, a procedure call to the program module verifier that is logically positioned in the second program module so as to be executed prior to completion of execution of the second program module's executable procedure, and instructions preventing completion of execution of that executable procedure when the program module verifier returns a verification denial with respect to the first program module.

The program module verifier responds to procedure calls by verifying the authenticity of any specified program module and by returning a verification confirmation or denial in response to each such procedure call. More specifically, in a preferred embodiment, the program verifier module includes instructions for responding to a procedure call requesting verification of a specified program module by (A) decoding a digital signature in the specified program module with a corresponding decoding key, (B) generating a message digest of at least a portion the specified program module in accordance with a message digest function, (C) returning a verification confirmation when the decoded digital signature matches the message digest, and (D) returning a verification denial when the decoded digital signature does not match the message digest.

In the preferred embodiment, when the program module verifier fails to verify the authenticity of the second program module, the first program module throws an exception and aborts its execution. Similarly, when the program module verifier fails to verify the authenticity of the first program module, the second program module throws an exception and aborts its execution.

BRIEF DESCRIPTION OF THE DRAWINGS

Examples of the invention will now be described in conjunction with the drawings, in which:

Fig. 1 is a block diagram of a computer system embodying the present invention.

Fig. 2 is a "time line" representation of how a typical procedure call is performed using the preferred embodiment of the present invention.

Fig. 3 is a flow chart of the method used in a preferred embodiment for two linked software modules to verify each other's authenticity.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring to Fig. 1, there is shown a computer system 100. While the computer 100 may be a desktop computer, such as a Sun workstation, IBM compatible computer, or Macintosh computer, virtually any type of computer could be used. The computer 100 includes a CPU 102, a user interface 104, and memory 106. Memory 106 includes primary random access memory (RAM) as well as secondary memory, typically one or more magnetic or optical disks. The memory 106 stores an operating system 110, a program module or object authenticity verifier 112, and a set of application program object instances 114, 116, 118, 120, also called program modules or application program modules.

As shown in Fig. 1, in a preferred embodiment of the invention each application program object instance includes an object header 122, at least one digital signature 124, at least one embedded public encryption key 126 and a main application procedure 128 (often called a method). Each method or procedure 128 includes at least one verifier procedure call instruction 130 and instructions 132 for responding to a verification denial message received in response to the verifier procedure call, such as instructions for aborting execution of the procedure. The main application A procedure (128-A) in the first program module furthermore includes a procedure call 134 to an executable procedure (e.g., the main application B procedure 128-B) in the second procedure module. The procedure call 130-A to the program module verifier is logically positioned in the first program module so as to be executed prior to execution of the procedure call 134 to the second program module.

The procedure call 130-B to the program module verifier is logically positioned in the second program module immediately after the entry point to each executable procedure 128-B in the second program module so as to be executed prior to execution of each such procedure 128-B. More generally, in other embodiments of the invention the procedure call 130-B to the program module verifier is logically positioned in the second program module (and more generally, in all program modules that will be called by other program modules) prior to the completion point in each executable procedure in the second program module so as prevent completion of the execution of each such procedure if verification of the calling program is denied by the verifier.

In a preferred embodiment of the present invention all the procedures in a designated group, such as all the procedures used by a particular top level application or a suite of top level applications, have the same embedded public key 126 and all are digitally signed using the same private encryption key, for example using the RSA encryption methodology. However, in an alternate embodiment, different procedures and subgroups of procedures are signed with different private keys. In the alternate embodiment, the procedure modules that include procedure calls have embedded public keys for verification of the procedures that they can call, and all procedure modules that can be called by other procedures include public keys for verification of the calling procedures.

Fig. 2 is a "time line" representation of how a typical procedure call is performed using the preferred embodiment of the present invention. In Fig.2 earlier events are shown at higher vertical positions than later events. Fig. 3 is a flow chart of the steps involved in the performance of a procedure call.

Referring to Figs. 2 and 3, an executable procedure (e.g., the "main application A procedure" 128-A in Figure 1) in program module A begins execution (step 200). For the purposes of this discussion, the procedure in program module A that is being executed will be called "procedure A" and the procedure that it is attempting to call in program module B will be called "procedure B".

Prior to making a procedure call to an executable procedure in program module B (step 220), procedure A makes a procedure call to the verifier to request verification of the authenticity of program module B (step 202). The verifier then attempts to verify the authenticity of program module B and sends a return value to procedure A to indicate whether or not the verification of program module B was successful (step 204).

More specifically, the verifier, which is preferably a distinct trusted object (or alternately a trusted system service procedure) receives the request message from procedure A (step 206), and decodes (step 208) a digital signature embedded in program module B using a public key provided by the calling procedure (i.e., procedure A). The public key provided by calling procedure A to the verifier is the "group" public key 126-A embedded in program module A.

In the preferred embodiment, the digital signature of a program module is generated by computing the message digest of the program module, supplementing the message digest with a hash function identifier to indicate the type of hash function used to generate the message digest, encrypting the resulting value with a private key using the RSA encryption methodology, and then supplementing that encrypted value with a clear text identifier of the source (i.e., author or distributor) of the program module:

$$\text{MD}_B = \text{HashFunction(Program Module B)}$$

$$\text{Digital Signature}_B = \text{Encrypt(MD}_B + \text{HashFunction ID, PrivateKey)}$$
$$+ \text{ClearText ID of Program Module B's Source}$$

Therefore to decode the digital signature of program module B, the verifier (A) removes the clear text ID from the digital signature, and then (B) decodes the remaining portion of the digital signature with the public key to generate a signature based message digest DS-MD_B and a hash function ID.

$$\text{DS-MD}_B + \text{HashFunction ID}$$
$$= \text{Decode (Digital Signature}_B - \text{ClearText ID, PublicKey)}$$

Next, the verifier computes a message digest MD_B of at least a portion of program module B (step 210) using the hash function identified in the decoded digital signature. The hash function used to generate the message digest is typically a function, such as a CRC encoding function, that is known to generate distinct values for distinct program modules with extremely high probability. Many hash functions suitable for generating a message digest are known to those skilled in the art.

The verifier then compares the computed message digest MD_B with the message digest DS-MD_B in the decoded digital signature (step 212) and returns a verification confirmation message to the calling procedure (step 214) if the two message digests match and returns a verification denial message (step 216) if the two message digests do not match.

In one preferred embodiment, there is a single digital signature for each program module, and the associated message digest is computed using a hash function of the entire contents of the program module. In other embodiments, the message digest may be based on only a portion of the program module. For instance, in a second preferred embodiment, each program module has two digital signatures: one for the methods portion of the program module and another for a data portion (if any) of the program module. When a program module has two digital signatures, both of the message digests derived by decoding the two digital signatures must match corresponding message digests computed by the verifier in order for the verifier to return a verification confirmation message. If the message digest in either decoded digital signature does not match the corresponding message digest computed by the verifier, the verifier returns a verification denial message.

If the verifier denies verification of program module B (step 216), procedure A "throws an exception" and then aborts (step 218). Throwing an exception generally causes the associated thread of execution to terminate and furthermore causes an exception handler procedure to be executed by the calling thread of execution so as to give the calling thread of execution the opportunity to analyze and otherwise respond to the failure of the called procedure (i.e., procedure A in this case).

Generally, the verifier will deny verification of a program module only if the program module has been corrupted, such as during installation or transmission from one computer to another, or deliberately tampered with. In normal operation, verification denials should be unusual events.

If the verifier confirms verification of program module B (step 214), procedure A then goes ahead with making a procedure call to procedure B in program module B (step 220). In the preferred embodiment, one of the very first things that procedure B does upon receiving a procedure call is to make a procedure call to the verifier (step 222), sending it a request to verify the authenticity of the calling program module (i.e., program module A in this case).

The verifier then attempts to verify the authenticity of program module A and sends a return value to procedure B to indicate whether or not the verification of program module A was successful (step 230).

More specifically, the verifier receives the request message from procedure B (step 232), and decodes (step 234) a digital signature embedded in program module A using a public key provided by procedure B. The public key provided by procedure B to the verifier is the "group" public key 126-B embedded in program module B.

As explained above, the digital signature of program module A is generated by computing the message digest of program module A, supplementing the message digest with a hash function identifier to indicate the type of hash function used to generate the message digest, encrypting the resulting value with a private key using the RSA encryption methodology, and then supplementing that encrypted value with a clear text identifier of the source (i.e., author or distributor) of the program module:

$$MD_A = \text{HashFunction(Program Module A)}$$

$$\text{Digital Signature}_A = \text{Encrypt}(MD_A + \text{HashFunction ID, PrivateKey})$$

$$+ \text{ClearText ID of Program Module A's Source}$$

Therefore to decode the digital signature of program module A, the verifier (A) removes the clear text ID from the digital signature, and then (B) decodes the remaining portion of the digital signature to generate a signature based message digest DS-MD_A and a hash function ID.

$$\text{DS-}MD_A + \text{HashFunction ID}$$

$$= \text{Decode (Digital Signature}_A - \text{ClearText ID, PublicKey)}$$

Next, the verifier computes a message digest MD_A of at least a portion of program module A (step 236) using the hash function identified in the decoded digital signature.

The verifier then compares the computed message digest MD_A with the message digest DS-MD_A in the decoded digital signature (step 238) and returns a verification confirmation message to the calling procedure (step 240) if the two message digests match and returns a verification denial message (step 242) if the two message digests do not match.

If the verifier denies verification of program module A (step 216), procedure B "throws an exception" and then aborts (step 244).

If the verifier confirms verification of program module A (step 240), procedure B is executed to completion (step 250) and the result generated by executing procedure B is returned to procedure A (step 252). Finally, procedure A completes its execution using the results received from procedure B (step 254).

ALTERNATE EMBODIMENTS

In some alternate embodiments only a portion of the program modules in a group of program modules contain "sensitive" algorithms or which otherwise are more important to authenticate than the other program modules. For instance, in a first alternate embodiment the distributor of a set of program modules (herein called the "full set of program modules") might want to ensure that a small number of program modules (herein called the "restricted subset of program modules") in the group are used only with the other program modules in the group, but might want to allow the remaining program modules to be used by the licensee freely, even with program modules outside the group. In this embodiment, only the restricted set of program modules include procedure calls to the verifier module, logically located immediately after the entry points of those program modules. These entry point procedure calls to the verifier are used to verify the authenticity of the calling program module, and upon verification that the calling program module is part of the authenticated group, the called program module performs the computation requested by the calling program module.

In a second alternate embodiment, the distributor of a set of program procedures is not concerned with restricting the use of a set of "restricted program modules," and is instead concerned that all calling procedures attempting to use the services of the restricted program modules in fact get the services of authentic versions of the restricted program modules. In this embodiment, all procedures making procedure calls to the restricted program modules include procedure calls to the verifier module logically located immediately prior to the procedure calls to the restricted program modules. These verifier procedure calls are used to verify the authenticity of the restricted program modules. However, in this embodiment the procedures in the restricted program modules do not contain verifier procedure calls to authenticate the calling program modules. Upon verification that the restricted program module to be called is authentic, the calling program module sends its procedure call to the authenticated restricted program module.

Claims

1. A computer system comprising:

(A) a program module verifier configured to respond to procedure calls to said program module verifier by verifying authenticity of any specified program module and by returning a verification confirmation or denial in response to each such procedure call;
(B) a first program module, and
(C) a second program module;

one of said first and second program modules including a procedure call to the other of said first and second program modules;
at least one of said first and second program modules including:

a procedure call to said program module verifier to verify authenticity of the other of said first and second program modules, and
instructions for aborting execution of said one program module when said procedure call to said program module verifier results in a verification denial being returned by said program module verifier.

2. The computer system of claim 1,

said first program module including a first digital signature and a first executable procedure;
said second program module including a second digital signature and a second executable procedure;
said program verifier module including instructions for responding to a procedure call requesting verification of a specified one of said first and second program modules by (A1) decoding said digital signature in said specified program module with a corresponding decoding key, (A2) generating a message digest of at least a portion said specified program module in accordance with a predefined message digest function, (A3) returning a verification confirmation when said decoded digital signature matches said message digest, and (A4) returning a verification denial when said decoded digital signature does not match said message digest.

3. The computer system of claim 1,

said first program module including a procedure call to said second program module;
said second program module including:

(C1) an executable procedure to be performed in response to said procedure call to said second program module;
(C2) a procedure call to said program module verifier logically positioned in said second program module so as to be executed prior to execution of said executable procedure; and
(C3) instructions preventing execution of said executable procedure when said procedure call to said program module verifier results in a verification denial being returned by said program module verifier.

4. The computer system of claim 3, wherein said instructions preventing completion of execution of said second executable procedure include instructions for aborting execution of said second program module when said procedure call to said program module verifier results in a verification denial being returned by said program module verifier.

5. The computer system of claim 1, 2, 3 or 4,
said first program module including:

a procedure call to said second program module;
a procedure call to said program module verifier logically positioned in said first program module so as to be executed prior to execution of said procedure call to said second program module; and
instructions preventing execution of said procedure call to said second program module when said procedure

call to said program module verifier results in a verification denial being returned by said program module verifier.

6. A method of linking program modules, comprising the steps of:

(A) prior to making a procedure call from a first program module to a second program module, verifying said second program module's authenticity;
(B) upon verifying said second program module's authenticity, making said procedure call from said first program module to said second program module; and
(C) upon failing to verify said first program module's authenticity, preventing said procedure call from said first program module to said second program module.

7. The method of claim 6, further including:

(D) prior to completing executing a procedure in said second program module in response to said procedure call by said first program module, verifying said first program module's authenticity;
(E) upon verifying said first program module's authenticity, completing executing said procedure in said second program module to generate a result and returning said result to said first program procedure; and
(F) upon failing to verify said first program module's authenticity, preventing completion of execution of said procedure in said second program module.

8. The method of claim 7,
said step (D) including decoding said first digital signature in said first program module with a corresponding decoding key, generating a message digest of at least a portion said first program module in accordance with said predefined message digest function, verifying the authenticity of said first program module when said decoded digital signature matches said message digest, and denying verification of the authenticity of said first program module when said decoded digital signature when said decoded digital signature does not match said message digest.

9. The method of claim 6, 7 or 8, wherein step (C) includes aborting execution of said first program module.

10. The method of claim 6, 7 or 8, wherein said first program module includes a first digital signature and said second program module includes a second digital signature;
said step (A) including decoding said second digital signature in said second program module with a corresponding decoding key, generating a message digest of at least a portion said second program module in accordance with a predefined message digest function, verifying the authenticity of said second program module when said decoded digital signature matches said message digest, and denying verification of the authenticity of said second program module when said decoded digital signature when said decoded digital signature does not match said message digest.

11. The method of claim 6, 7 or 8,
said step (A) including making a procedure call to a trusted program module verifier, said program module verifier responding to said procedure call by verifying authenticity of said second program module and by returning a verification confirmation or denial in response to said procedure call.

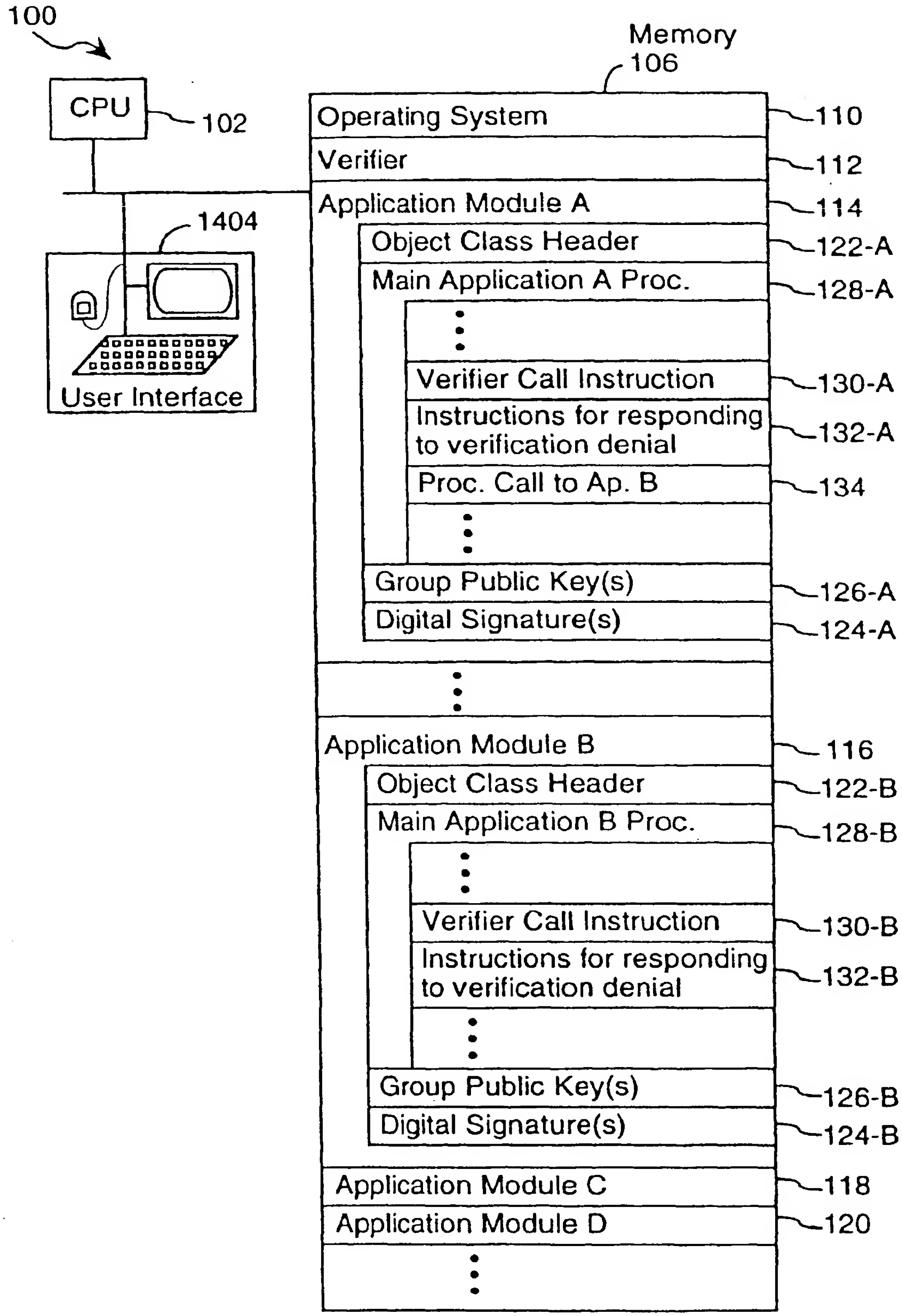
100
CPU
102
Memory
106
1404
User Interface
Operating System
110
Verifier
112
Application Module A
114
Object Class Header
122-A
Main Application A Proc.
128-A
Verifier Call Instruction
130-A
Instructions for responding to verification denial
132-A
Proc. Call to Ap. B
134
Group Public Key(s)
126-A
Digital Signature(s)
124-A
Application Module B
116
Object Class Header
122-B
Main Application B Proc.
128-B
Verifier Call Instruction
130-B
Instructions for responding to verification denial
132-B
Group Public Key(s)
126-B
Digital Signature(s)
124-B
Application Module C
118
Application Module D
120

FIGURE 1

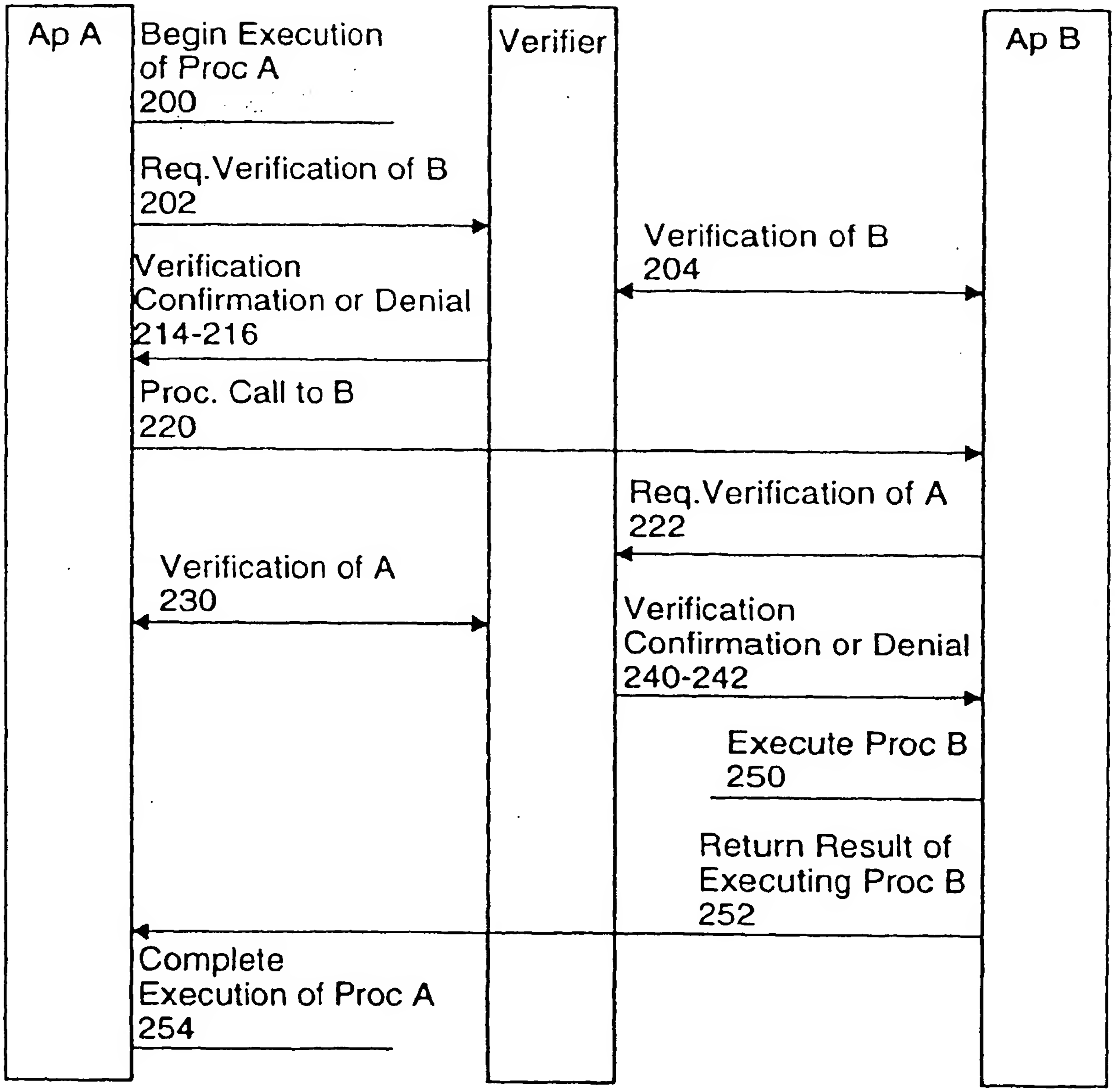
Ap A
Verifier
Ap B
Begin Execution of Proc A 200
Req.Verification of B 202
Verification of B 204
Verification Confirmation or Denial 214-216
Proc. Call to B 220
Req.Verification of A 222
Verification of A 230
Verification Confirmation or Denial 240-242
Execute Proc B 250
Return Result of Executing Proc B 252
Complete Execution of Proc A 254

FIGURE 2

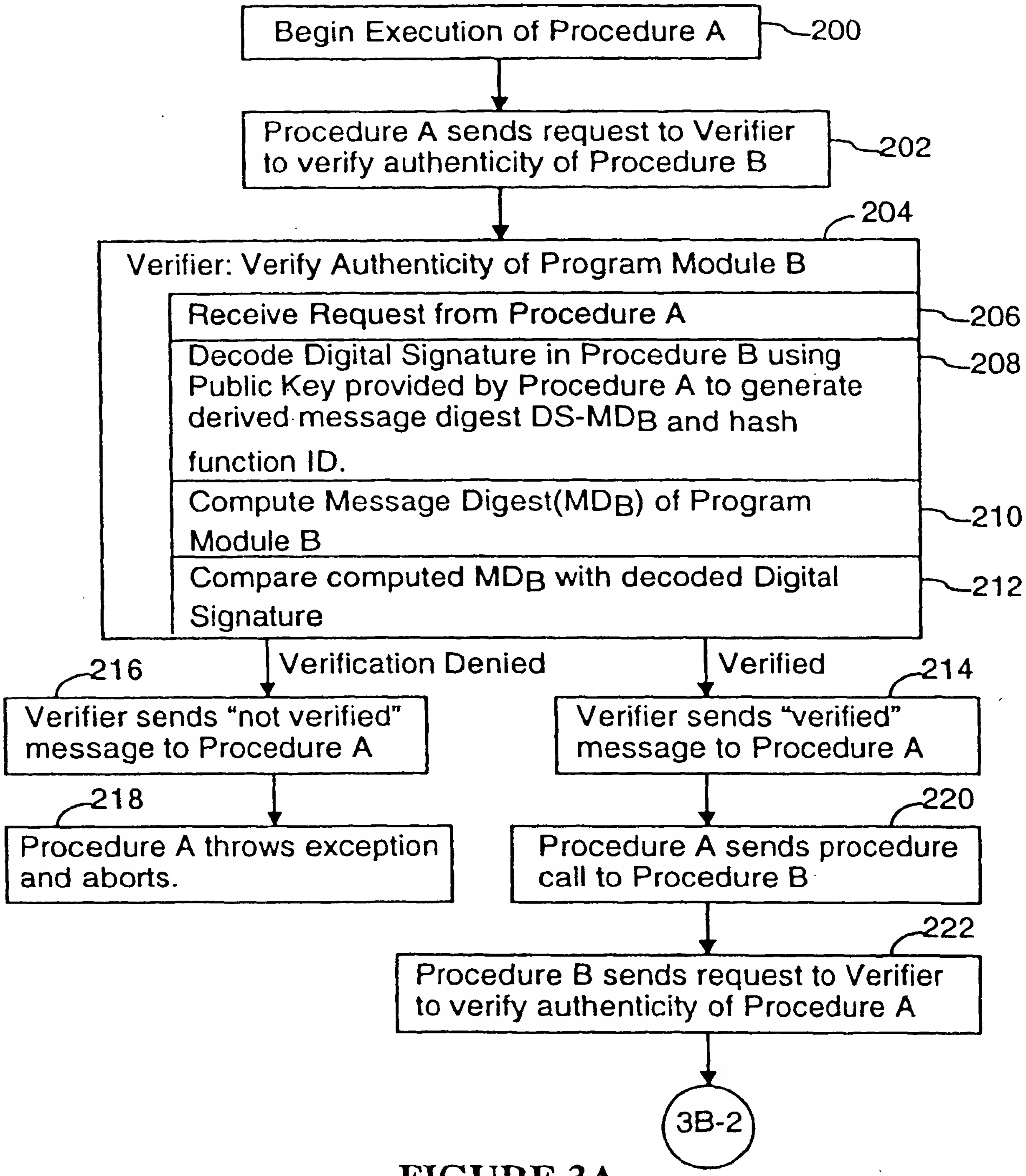
Begin Execution of Procedure A
200
Procedure A sends request to Verifier to verify authenticity of Procedure B
202
204
Verifier: Verify Authenticity of Program Module B
Receive Request from Procedure A
206
Decode Digital Signature in Procedure B using Public Key provided by Procedure A to generate derived message digest DS-MDB and hash function ID.
208
Compute Message Digest(MDB) of Program Module B
210
Compare computed MDB with decoded Digital Signature
212
Verification Denied
Verified
216
214
Verifier sends "not verified" message to Procedure A
Verifier sends "verified" message to Procedure A
218
220
Procedure A throws exception and aborts.
Procedure A sends procedure call to Procedure B
222
Procedure B sends request to Verifier to verify authenticity of Procedure A
3B-2

FIGURE 3A

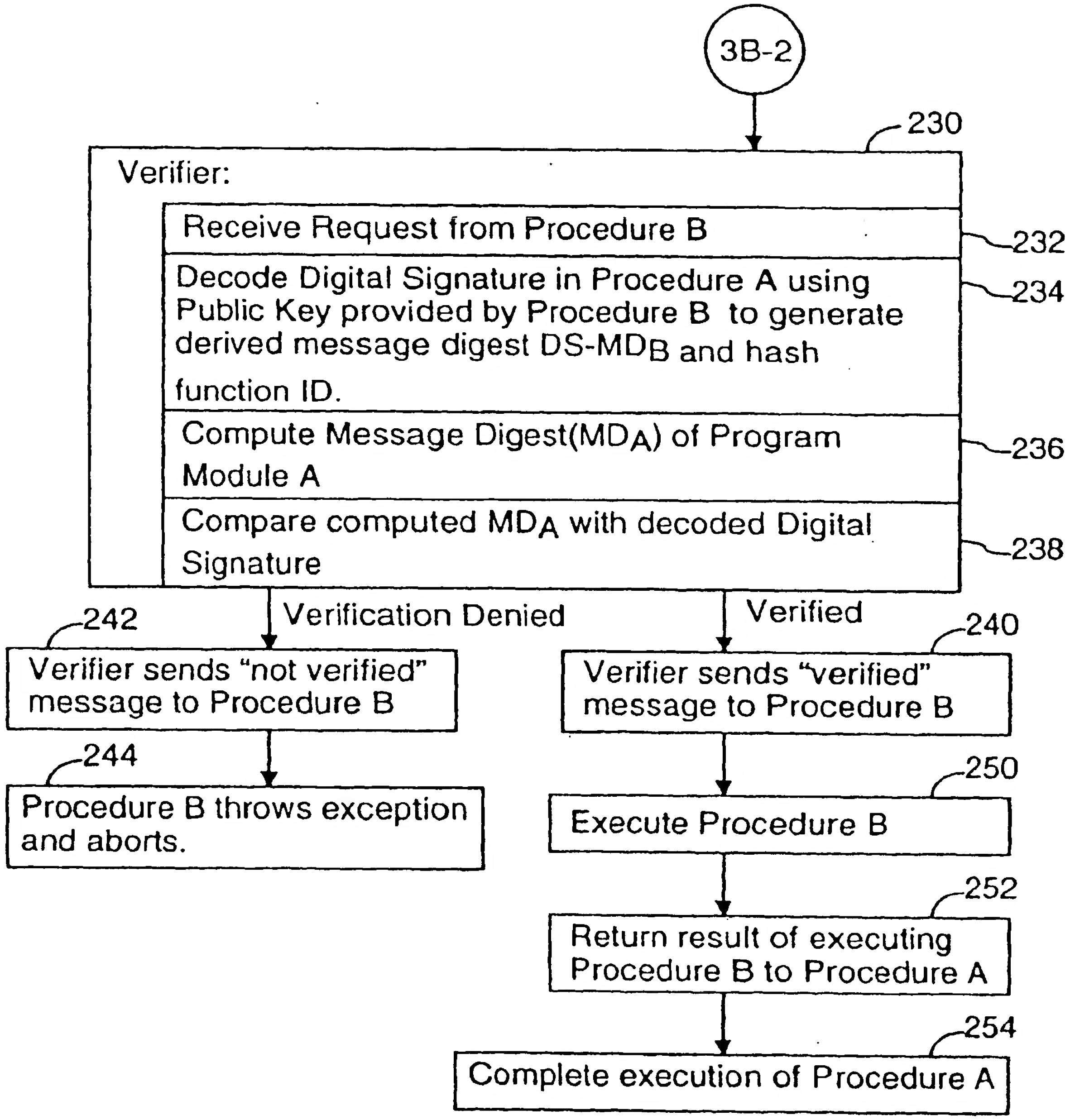
3B-2
230
Verifier:
Receive Request from Procedure B
232
Decode Digital Signature in Procedure A using Public Key provided by Procedure B to generate derived message digest DS-MD$_B$ and hash function ID.
234
Compute Message Digest(MD$_A$) of Program Module A
236
Compare computed MD$_A$ with decoded Digital Signature
238
Verification Denied
Verified
242
Verifier sends "not verified" message to Procedure B
240
Verifier sends "verified" message to Procedure B
244
Procedure B throws exception and aborts.
250
Execute Procedure B
252
Return result of executing Procedure B to Procedure A
254
Complete execution of Procedure A

FIGURE 3B

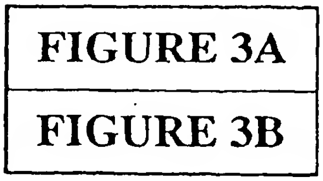

FIGURE 3

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 770 957 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
15.03.2000 Bulletin 2000/11

(43) Date of publication A2:
02.05.1997 Bulletin 1997/18

(21) Application number: 96307347.3

(22) Date of filing: 09.10.1996

(51) Int Cl.7: G06F 9/445, G06F 1/00,
G06F 9/40

(84) Designated Contracting States:
DE FR GB NL SE

(30) Priority: 26.10.1995 US 547720

(71) Applicant: SUN MICROSYSTEMS, INC.
Mountain View, California 94043-1100 (US)

(72) Inventor: McManis, Charles E.
Sunnyvale, California 94087 (US)

(74) Representative:
Cross, Rupert Edward Blount et al
BOULT WADE TENNANT,
27 Furnival Street
London EC4A 1PQ (GB)

(54) System and method for protecting use of dynamically linked executable modules

(57) A computer system has a program module verifier and at least first and second program modules. Each program module includes a digital signature and an executable procedure. The first program module furthermore includes a procedure call to the second procedure module, a procedure call to the program module verifier that is logically positioned in the first program module so as to be executed prior to execution of the procedure call to the second program module, and instructions preventing execution of the procedure call to the second program module when the procedure call to the program module verifier results in a verification denial being returned by the program module verifier. The second program module includes an executable procedure to be performed in response to the procedure call by the first program module to the second program module, a procedure call to the program module verifier that is logically positioned in the second program module so as to be executed prior to completion of execution of the second program module's executable procedure, and instructions preventing completion of execution of that executable procedure when the program module verifier returns a verification denial with respect to the first program module. The program module verifier responds to procedure calls by verifying the authenticity of any specified program module and by returning a verification confirmation or denial. When the program module verifier fails to verify the authenticity of a program module, the calling program module throws an exception and aborts its execution.

Printed by Jouve, 75001 PARIS (FR)

European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 30 7347

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	US 5 339 403 A (PARKER THOMAS A) 16 August 1994 (1994-08-16) * figures 1-3 * * column 1, line 67 - column 5, line 58 * ---	1,3-6,9,11	G06F9/445 G06F1/00 G06F9/40
A	US 5 349 642 A (KINGDON KEVIN) 20 September 1994 (1994-09-20) * figures 1-3,7 * * column 5, line 24 - column 7, line 9 * * column 9, line 33 - line 62 * ---	1,2,6-8,10	
X	GOSLING J ET AL: "THE JAVA LANGUAGE ENVIRONMENT. A WHITE PAPER" SUN DELIVERS JAVA WORKSHOP,XX,XX, page 1,4-85 XP002042922 * page 58, line 1 - page 60, line 3 * -----	1,3-6	
			TECHNICAL FIELDS SEARCHED (Int.Cl.6)
			G06F H04L

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	21 January 2000	Weiss, P

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 96 30 7347

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

21-01-2000

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5339403 A	16-08-1994	AU 634653 B	25-02-1993
		AU 7620991 A	14-11-1991
		DE 69130461 D	17-12-1998
		DE 69130461 T	10-06-1999
		EP 0456386 A	13-11-1991
US 5349642 A	20-09-1994	AT 185661 T	15-10-1999
		AU 673393 B	07-11-1996
		AU 5457894 A	24-05-1994
		BR 9307360 A	01-06-1999
		CA 2148105 A,C	11-05-1994
		DE 69326775 D	18-11-1999
		EP 0667998 A	23-08-1995
		JP 8507416 T	06-08-1996
		WO 9410778 A	11-05-1994

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82